The Triad System: the Design of a Distributed, Real-Time, Trusted System

E. John Sebes
Pierre X. Pasturel

Trusted Information Systems, Inc.
444 Castro Street, Suite 800
Mountain View, CA 94041
415/962-8885
{ejs,pasturel}@ba.tis.com

Terry C. Vickers Benzel
Dennis Hollingworth
Eve L. Cohen Peter Wang
Trusted Information Systems, Inc.
11340 W. Olympic Blvd. Suite 265
Los Angeles, CA 90064
310/477-5828
{tcvb,holly,elc,pwang}@la.tis.com

Michael Barnett
David M. Gallon
Roman Zacjew
Locus Computing Corp.
5910 Pacific Center Blvd.
San Diego, CA 92121
619/546-9500
{mbarnett,dmg,roman}@locus.com

Abstract

Triad is a prototype distributed operating system that provides a high-assurance trusted system platform for real-time distributed applications that process information of various classifactions. Triad provides an extensible software architecture in which applications can make uniform use of, and also build upon its system capabilities. This paper describes the design of the Triad system, the framework for applications using the system, an example application scenario, and selected specific distributed processing techniques used by the Triad system.

1 Introduction

Triad is a prototype distributed operating system providing a high-assurance trusted system platform for real-time distributed applications that process information of various classifications. The Triad system development has merged and advanced three areas of advanced operating system functionality: multilevel security, real-time, and distributed processing. This paper describes the design of the Triad system and an example of typical application-level processing using all of the system's primary capabilities.

In addition, two critical aspects of the design are explored: extensibility and scalability. Extensibility results from Triad's microkernel basis and its client/server architecture. Triad system software is implemented in servers running on the microkernel. Application servers can easily extend the server architecture so that application software can make uniform use of system and application capabilities. Scalability results from another critical feature of the system's design: its decomposition into a framework of discrete modules for separate distributed system capabilities. Within this framework the prototype has implemented each of these distributed system capabilities using techniques appropriate to real-time distributed military C^3I applications. Because of the modular design, alternative techniques can be implemented without perturbing the system design. As a result, the system can be extended for systems with larger scales and/or different distribution requirements.

Before describing Triad's design in Section 3, we first describe the system's basic capabilities in Section 2. Section 6 provides an account of a typical application's execution and use of system capabilities. Finally, Section 7 describes the extensibility of the system, including sample extensions. Support for such extensions is the focus of ongoing development of the Triad system.

2 System Capabilities

The three primary capabilities of Triad are distributed system service, support for real-time applications, and the uniform enforcement of a security policy throughout a Triad distributed system. A key part of Triad's architecture (described in [1]) is its basis on Trusted Mach (TMach) [1] [3], a high-assurance system base designed to meet B3 requirements for security functionality and assurance.[2] Triad's security capabilities derive in large part from this trusted system base. Triad extends the TMach system so that its security functionality is uniformly and consistently performed throughout a group of hosts composing a Triad system.

This approach—to start with a high-assurance trusted system base and add distributed real-time functionality—was motivated by our belief that it would be more likely to yield a high-assurance system than would an alternative strategy of over-laying trust on an untrusted system with distributed real-time functionality. This approach applies particularly well to TMach, which was designed to be extensible, as a result of its multi-server architecture and Mach [10] microkernel base.

Triad's real-time design relies on the real-time notion of *timeliness*, i.e., the restriction that some computations complete within a finite time window. The base of Triad's real-time capabilities is its scheduling subsystem, which allows threads of execution to set deadlines and which uses deadlines to determine which threads to schedule. Such time-bound processing is ensured by deadline-based scheduling that is uni-

[1] Trusted Mach and TMach are Registered Trademarks of Trusted Information Systems, Incorporated.

[2] TMach is currently under evaluation for a B3 TCSEC rating and an E5/F-B3 ITSEC rating.

and to provide a basis for controlled sharing of information between organizations. A number of computer operating system security controls [2, 3, 4, 7, 10] have been proposed, but most of these techniques either map poorly to commercial requirements or impose excessive administrative overheads. Domain and Type Enforcement (DTE) [1] is a relatively recent operating system access control mechanism that holds promise to provide needed security flexibility and strength while controlling administrative costs. We believe that a fundamental question for practical commercial security is whether strong-but-flexible access controls, such as DTE, can be combined with firewall defenses in a way that preserves the practicality of firewalls while substantially improving security.

This paper reports on our experience with a DTE-enhanced firewall prototype. Our preliminary results indicate that DTE access controls *can* be integrated with firewalls to cost-effectively control resources shared through firewalls and to protect enterprise resources from possibly malicious insider programs. After reviewing the primary concepts of DTE, this paper presents the design of a DTE firewall and the mechanisms it uses to control imported and exported services. This paper next presents our informal evaluation of DTE firewall security, functionality, compatibility, and performance characteristics. Finally, this paper reviews related work and future directions, and presents conclusions.

2. DTE review

DTE [1, 15, 18] is an enhanced form of type enforcement [4, 13, 14], which is a table-oriented mandatory access control mechanism. DTE has three main advantages over type enforcement. First, the security policy is specified in a high level language that reduces the burden of expressing, verifying, and maintaining security rules. Second, security attributes on objects are implicit, thereby allowing a file hierarchy to be typed concisely. Finally, DTE provides mechanisms for backward compatibility with existing software and with systems not running DTE.

As with a number of other access control mechanisms [2, 3, 4, 7, 11], DTE considers a system to be split logically into two categories: passive entities (e.g., files or network packets) and active entities (usually processes). A type is associated with each passive entity, or object; a domain and a DTE-protected user identifier (unchangeable even by the *root* user) is associated with each active entity, or subject. Using a language-based specification, DTE expresses allowed interactions between subjects and objects. Access control decisions are made by consulting a DTE database consisting of the "compiled" specification to determine if the subject's domain has the requested access (e.g., read or write) to the object's type. DTE also expresses allowed interactions between subjects. Access control logic consults the DTE database to determine if a subject A's domain has the requested access (e.g., execute or kill) to a subject B's domain.

To extend DTE protection across networks, DTE treats each network packet as an object with three associated attributes (carried in the IP option space of each datagram): the DTE type of the information, the domain (source domain) of the source process, and the DTE-protected UID of the source process. A process can send or receive a message object only if the process's domain has the appropriate access to the DTE type of the message. Communication with non-DTE systems also is mediated: when a message originates from a non-DTE system, the receiving DTE system assigns a type (and domain) to the message.[2] Similarly, a DTE system mediates a message before sending it to a non-DTE system to ensure that the domain associated with the non-DTE system can read the messages. If the associated domain cannot read the message's type, the message is not sent.

In our prototype, UNIX[3] kernel changes to enforce DTE mediation are localized to a relatively small subsystem; all system calls that represent process accesses to other processes or to objects are passed through this subsystem.[4] During the initialization phase of a DTE kernel (i.e., at boot-time), the DTE subsystem reads a security specification written in the DTE Language (DTEL) from the boot device, parses the specification into access control data structures, and then mediates the system calls of all processes (including the first system process) according to the specification, as described above.

3. DTE Firewall overview

A DTE firewall is an enhanced application gateway firewall. It runs application gateways in controlling

[2] A DTE policy currently associates attributes with packets received from non-DTE systems based on source IP address. While IP addresses can be spoofed in IPv4, the approach is adequate for prototyping. Stronger mechanisms could be provided using IP-layer cryptography, such as that proposed for IPv6.

[3] UNIX is a registered trademark in the United States and other countries, licensed exclusively through X/Open Company Ltd.

[4] Our prototype is based on BSD/OS 2.0 (and recently 2.1), a widely available PC UNIX. Excluding comments, DTE enhancements to the kernel represent approximately 14,000 lines of code, and DTE enhancements to the firewall proxies represent approximately 700 hundred lines of code.

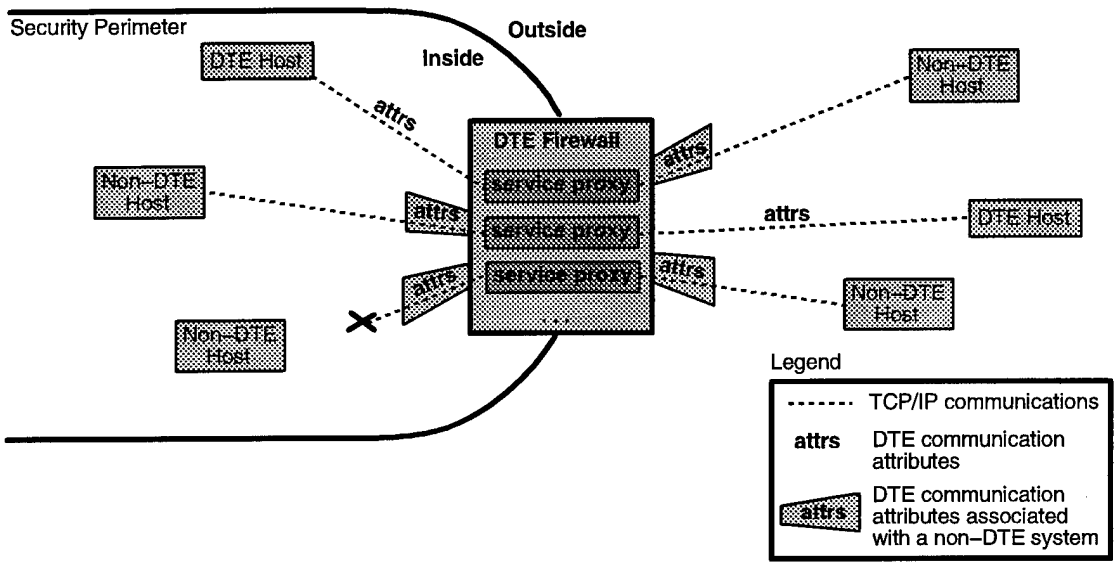

Figure 1. DTE Firewall Concept of Operations

DTE domains and mediates network communications based on DTE security attributes.

Figure 1 shows an overview of a network enclave protected by a DTE firewall. Some hosts behind the firewall are running DTE; there may or may not be DTE hosts outside the perimeter. As with "normal" firewalls, a DTE firewall intercepts and mediates all network traffic between internal and external hosts. Based on the DTE policy, the DTE firewall associates domains with non-DTE hosts and associates DTE communication attributes with data received from non-DTE hosts. As described in section 2, the DTE firewall and DTE hosts use the communication attributes to ensure that network messages are received only by domains that can appropriately control the type of data carried by the messages. If the end host is a DTE system, it carries the responsibility for confining network services: in this case, the DTE firewall's role is to coordinate the endpoints' security contexts by passing along the DTE communication attributes. In the case of non-DTE hosts, the DTE firewall performs access control on behalf of non-DTE hosts (interior or exterior) by mediating the network messages sent to each non-DTE host based on whether the host's associated domain grants access to the DTE attributes associated with the messages. By coordinating DTE policies between DTE endpoints (and on behalf of non-DTE endpoints), the DTE firewall is positioned both to protect and control exported services and to confine network clients that import services.

3.1. Controlling exported services

Figure 2 illustrates our general strategy for exporting services safely: run network server applications either on DTE firewalls or on DTE hosts behind the security perimeter and use DTE to control access to local resources. In figure 2, a DTE server system hosting a network server application (*service*) communicates through a DTE firewall to respond to service requests from a non-DTE external host (*Client System*). Since the client shown is not running DTE, the firewall associates type t and domain sd with messages received from it. The DTE firewall relays the DTE communication attributes to the server: these attributes establish DTE access controls consistent with the level of trust the firewall places in the client.

When a client attempts to initiate a connection with the server, the *inetd* daemon on the firewall runs the *netacl* program, which determines whether communication is allowed between the client and server hosts[5] and executes the *proxy* application for the specified protocol. The *netacl* program executes the *proxy* in the *proxy_d* domain, which is specific to the protocol (e.g., the HTTP proxy runs in the domain *fw_http_d*) and has DTE access permissions sufficient to pass DTE communication attributes between the client and the server.[6]

[5]This is standard application gateway firewall functionality; the check is based on IP addresses or host names.

[6]This paper assumes a homogeneous DTE policy (having the same domain and type definitions) for all hosts; future work will introduce interactions between hosts running different policies

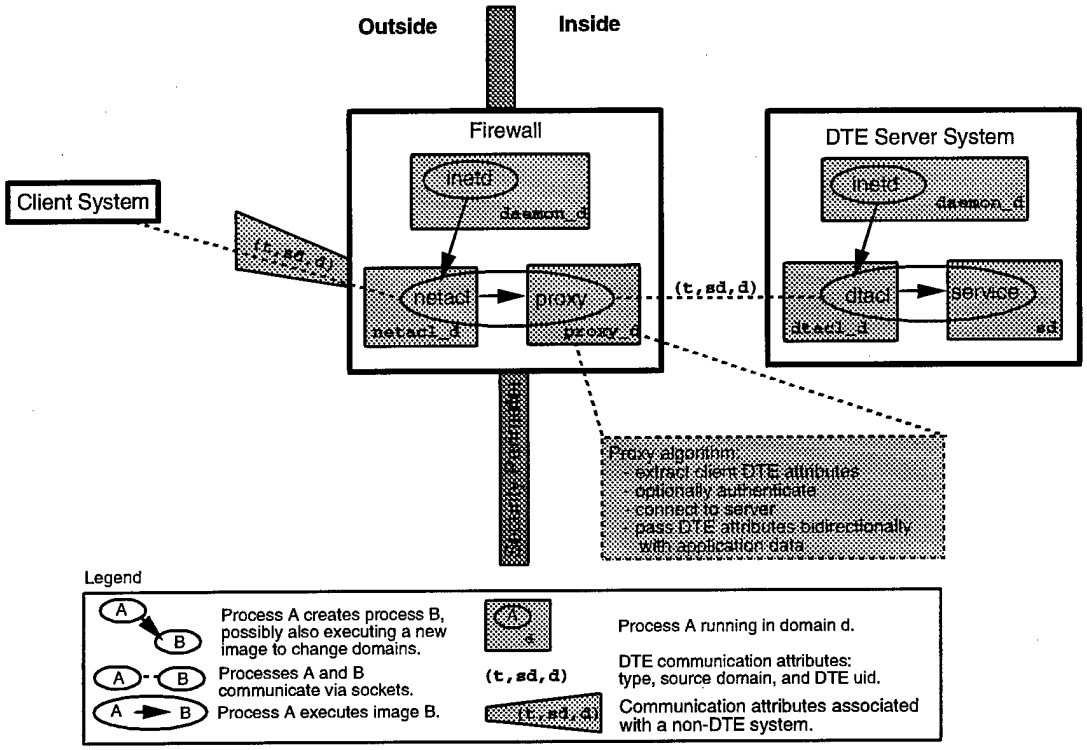

Figure 2. Exporting Services Safely

The *proxy_d* domain can be configured to control which clients may use the service.

At a high level, the *proxy*'s algorithm is simple:

1. **Extract client attributes** The DTE kernel's socket abstraction provides an interface to retrieve the client attributes *(t,sd,d)* each time data is read from the client socket. Because the attributes are carried in each IP message, they are available for both connection-oriented (e.g., TCP) and connectionless (e.g., UDP) protocols.

2. **Optionally authenticate** If the client is not a DTE system, the proxy authenticates the client using the method specified in the firewall's configuration. If the client is a DTE system, the proxy may choose to trust the DTE UID (*d*) in the communication attributes, which indicates the user's identity as authenticated by the client system. Using a DTE client's authentication attribute increases usability and performance by relieving the client of the need to authenticate for each service initiation (this has resulted in DTE firewall performance *increases* for some test cases).

3. **Connect to server** The proxy connects to the DTE server system, passing the DTE attributes of the client.

4. **Pass data and DTE attributes bidirectionally** The DTE-enhanced proxy scans and copies data as in a conventional firewall except that it also passes the DTE communication attributes through to each endpoint. Also, for each message-receive and message-send operation, the proxy's domain is mediated (by the firewall's DTE kernel) with respect to the type of data being received or sent: the types of data allowed through the firewall therefore can be adjusted by adjusting the types of data that the proxies are allowed to read/write. Because DTE attributes are associated with remote non-DTE hosts by the firewall, the policy also can be adjusted so that some types of data are not sendable (or receivable) from specified hosts or networks. When the external host is running DTE, the control is more fine-grained and communications using specific types of data can be allowed or disallowed for individual domains (or processes) on the remote host.

and dynamic policy reconfiguration.

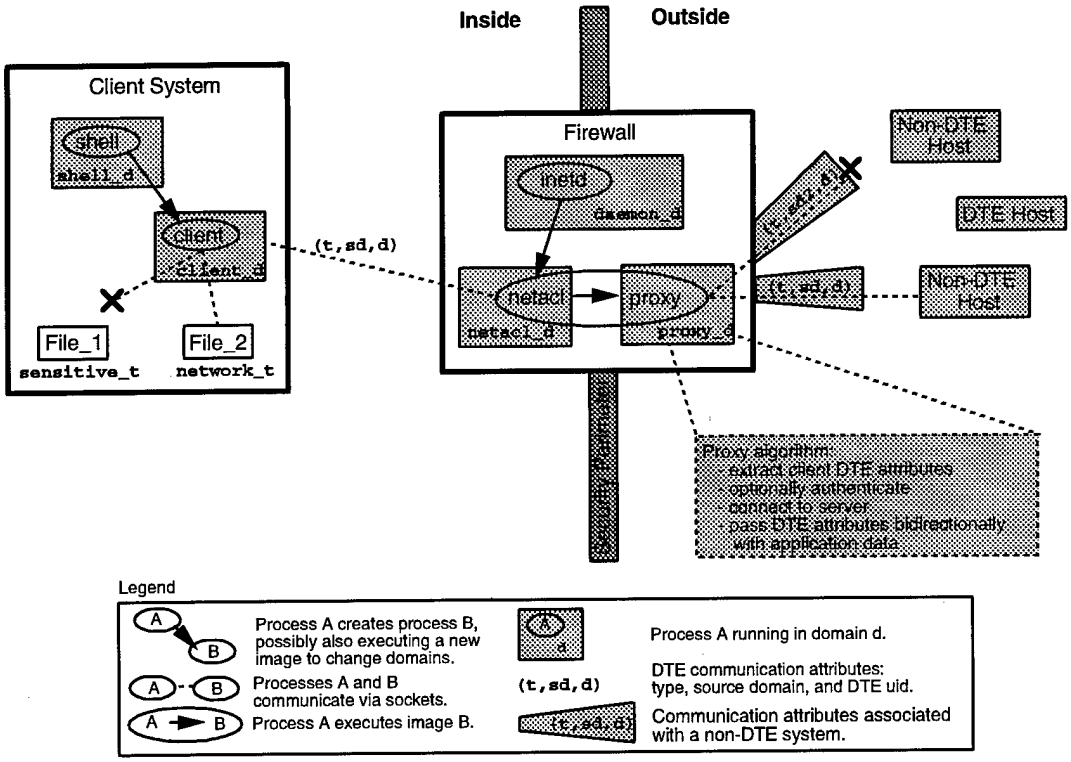

Figure 3. Importing Services Safely

As shown in figure 2, when the *proxy* application connects to the DTE server, the *inetd* program on the server runs the *dtacl* program. This program, like *netacl*, examines the DTE attributes sent in the connection request and then executes the server application *service* in domain *sd*, the domain of the client. By running the server application in the client's domain, the system ensures that the client and server run in a compatible security context: if the client is anonymous (e.g., anonymous FTP or HTTP), the server runs in a domain granting very limited access to data on the server system; if the client is a known quantity (perhaps administered by a business partner, or an authorized guest), the server runs in a domain granting access to more types of data.[7] However, for services that do not restart with each client request, such as NFS, a different strategy is needed. These services employ a kernel-supported "auxiliary" domain (i.e., the source domain of a received message) to further restrict the server when it is working on behalf of a particular client; DTE mediation then is performed using both domains. Because the service must ask the kernel to load and unload the auxiliary domain, this technique is available only to trusted servers.

The strategy of running the server in the client's domain does not necessarily result in least privilege, since the common domain may overstate access needs of the client or the server. However, the strategy is surprisingly simple and approximates least privilege (in our experience) closely enough to separate user roles and to protect the system. The DTE policy on the server host protects the host from the server application (using root confinement [18]) and also protects the server application from other programs that may run on the server host. Additionally, the types writable by the *sd* domain indelibly label data that originated from the exterior network. This labeling can alert users and programs that the data was received from the network and could be untrustworthy.

3.2. Controlling imported services

A DTE firewall controls imported services using the same mechanisms it uses to control exported services: the DTE firewall relays DTE communication attributes from an internal client system to potential server sys-

[7] The access level granted to domains depends upon the DTE policy and is user-configurable.

tems and performs mediation on behalf of non-DTE clients and servers. The overall effect of this mediation is to prevent a client from using a server unless either the server is running in a compatible domain or the server runs on a DTE host that is willing to start a copy of the server in the client's domain. As a consequence of these mechanisms, the same client operating in different domains may have differing levels of access to external services. Users on a DTE client, therefore, may choose restricted domains when accessing untrusted or unknown services (e.g., surfing the web), and choose more privileged domains when accessing important corporate data. Additionally, this constraint ensures, for example, that information cannot be accidentally sent to competitors (e.g., using mail) and that files cannot be accidentally imported from untrustworthy environments and executed.

In figure 3, a DTE client system runs a network program that communicates through the DTE firewall to access a service provided by a non-DTE external host. Figure 3 shows the user session running in domain *shell_d* and the client application running in domain *client_d*.[8] The domain *client_d* controls the client application in two important ways: 1) it prevents a successful attack on the client from damaging the client system, and 2) it labels data generated by the client application with a type that identifies the data's origin and indicates the amount of trust that can be placed in it. In addition, the DTE policy on the client system protects the client application from attacks by other running programs. This is particularly useful for preventing software of questionable quality (e.g., the most recent version of a free editor) from accessing the data streams of important client applications such as electronic commerce, banking, or Internet telephone programs.

As shown in figure 3, the client application communicates via sockets with the DTE firewall which then communicates with the external server. Since the serving host is not running DTE, the DTE firewall prevents communication with the serving host unless the DTE firewall associates a compatible domain with the host.

4. Network services evaluation

To assess the overall impact of DTE firewalls and selected hosts, we have evaluated informally the remote login (rlogin), TELNET, mail, FTP, NFS, and HTTP network services running through a DTE firewall. The evaluation considered the security, preservation of functionality, compatibility with non-DTE hosts, and performance of these services. For purposes of evaluation, we used the BSD/OS DTE prototype system and the TIS Firewall Toolkit (V1.3).

4.1. Security

Network attacks often exploit subtle weaknesses in programs that allow attackers to misuse program access rights, gain control over systems, steal or destroy data, and deny access to authorized users. The effectiveness of such attacks can be reduced if programs execute with the minimum access rights required to perform their functions. Our informal metric for evaluating the security of a DTE firewall, therefore, is the extent to which it restricts the access rights used by programs running on the firewall or on DTE systems that communicate with the firewall. We have identified three primary areas where program authorizations are reduced by DTE:

confined proxies A DTE firewall confines each network proxy in a separate domain. In general, the domains used to confine proxies prevent write access to system and administrative data and also prevent proxies from running other programs. This confinement protects the firewall from a possibly subverted proxy: for example, if the FTP proxy is compromised, it cannot alter system files.

Since proxies are trusted to propagate type labels of user data that passes through them, a faulty or subverted proxy may cause user data to be mislabeled, however this mislabeling can be limited: a proxy can attach a type label to an object only if the proxy's domain grants write access to the label. Consequently, the firewall's DTE policy can be configured to control the discretion afforded a proxy as well as to restrict which types of data are allowed to flow through different network service proxies.

protected servers on the firewall A DTE firewall can run network services in domains that protect their data and program files from network-based attack. By running services on the firewall (or, for load-sharing purposes, on companion DTE server hosts), the integrity of network services can be improved substantially even though the services are made available to the Internet. While running possibly vulnerable network services on a conventional firewall would pose an unacceptable security hazard, a DTE firewall can execute such services

[8] A DTE client system's DTE policy can be configured to give the user discretion in which domain the client application runs, to require that the client application run in the domain of the user's shell, or to specify automatically a domain in which the client application runs.

safely because their access rights are limited via DTE.

defense in depth As discussed in section 3.1, DTE firewalls coordinate security protections of DTE servers behind a security perimeter to restrict the access rights of programs that are run on behalf of external clients. When DTE servers are available, this strategy ensures that processing carried out on behalf of external clients is controlled in appropriate DTE domains that prevent clients from successfully tricking interior services into granting unauthorized access. This technique can be used to prevent unauthorized export of sensitive data through ubiquitous services, such as mail, and to export NFS hierarchies selectively to the Internet. Furthermore, this strategy typically does not rely on the correctness or security sophistication of services exported through the DTE firewall since the control is enforced by DTE on the serving hosts.[9]

The DTE firewall policy used to conduct our experiments consists of 105 lines of DTEL specification on the firewall and 122 lines of DTEL on supporting DTE hosts. The addition of a new network service or domain adds typically only 5 or 6 lines to these specifications. Additionally, small extensions to domains in the policy can be made that apply uniformly across the specification and expand or constrict access rights for all network services. As a result, DTE policy complexity can be controlled, resulting in enhanced assurance that protections are maintained as new network services are added. The combination of these techniques has significantly reduced the access rights used by software running on (or through) the firewall while allowing the software to function correctly. Based on our informal metric, we belive that these techniques have significantly increased network enclave security.

4.2. Functionality

For importing services, functionality is rarely affected. For services such as rlogin and TELNET, when the client is a DTE system, user authentication can be supplied automatically by the client DTE system and the proxy can accept and use this authentication instead of requiring additional authentication. Under some circumstances, this can increase usability; however, the DTE UIDs must be set up in the firewall configuration, adding a small amount of administrative overhead. Also, services such as HTTP and FTP can be made more widely available because of the reduced risk of malicious programs (e.g., applets).

Functionality for exported services has increased in several ways. With the additional security of running a server in a domain restricted according to trust level of the client, the server no longer needs to be located outside the firewall. Instead it can be on a system behind the firewall, or even on the firewall itself. Furthermore, with the server starting in the domain of the client, it is feasible to grant access to different classes of information based on that domain – something very useful for HTTP and FTP. In this way, for example, an anonymous FTP server can regulate access to files without resorting to multiple servers or hidden files. Also, a single HTTP server can provide sensitive data to users in trusted domains and general data to users in untrusted domains without fear of having the data compromised or altered. NFS requires slightly more administrative overhead, but becomes safely available through firewall security perimeters – something not possible before.

4.3. Compatibility

Overall, compatibility has been maintained: each service can interoperate either with DTE or non-DTE systems. Furthermore, the application-level proxies revert to standard Firewall Toolkit behavior when run on a non-DTE kernel. The use of IP options to carry DTE information removes the need for changing the protocols.

With the exception of the NFS server, which is kernel-resident in UNIX, few of the client or server applications have been changed to function with DTE firewalls. Mail final delivery agents were modified to be cognizant of the different types associated with users' mailboxes. Also, the rlogin server was modified to take advantage of DTE authentication mechanisms.

Some of the services running under DTE require changes in the administrative configuration. For example, external NFS clients must explicitly name the firewall host as the server whose file systems they wish to mount, since they cannot know the name of the server behind the firewall.

4.4. Performance

To evaluate the performance of DTE and DTE Firewalls, we have constructed a testbed consisting of three Pentium 166MHz machines on an isolated network running BSD/OS 2.0 and version "straw_19+" of the DTE prototype system.[10] We ran each test on a number of

[9]The exception is the NFS daemon, which for performance reasons is implemented as a trusted server.

[10]This is the 19th internal version of the BSD/OS-based DTE prototype with some performance enhancements incorporated;

Protocol		Data Transferred	Baseline (n,n,n)	Percentage Change (%) (n,y,n)	(n,y,y)	(y,y,n)	(y,y,y)
rlogin		0K	1.98	0	0	-15	-10
		200K	3.43	6	8	-16	-23
		500K	5.33	<1	<1	-14	-16
		5MB	32.21	<1	<1	-2	-2
TELNET		0K	6.79	<1	<1	-15	-12
		200K	7.79	<1	2	-10	-10
		500K	9.89	1	1	-9	-7
		5MB	41.36	<1	1	-2	-1
FTP		0K	1.98	9	13	6	11
		200K	3.98	4	6	4	5
		500K	4.59	3	4	3	4
		5MB	14.23	3	6	<1	3
		32MB	70.33	2	2	-1	<1
HTTP	Concurrency Level 4	1K	355.81	8	13	12	15
		50K	64.71	71	89	92	94
	Concurrency Level 8	1K	199.20	22	24	26	25
		50K	60.23	75	94	97	114

Figure 4. Raw Performance in Seconds and DTE Overheads

configurations where configuration is a triple (client, firewall, server) in which "y" indicates a system running DTE and "n" indicates a host not running DTE (so (n,y,n) is the configuration where only the firewall is running DTE). We did not measure the performance of mail since it is not interactive.

For rlogin, TELNET, and FTP, we used an Expect script to repeatedly authenticate and transfer various amounts of information through the service. Performance numbers were calculated by averaging results from 20 iterations of each test.

For HTTP, we used *ZeusBench*,[11] a standard benchmark that connects to the server via the HTTP proxy, retrieves a specified web page, and closes the connection. We varied the concurrency, the document length, and the number of requests (1000 requests for 1K documents, and, to save time, 32 requests for 50K documents).

As shown in figure 4, DTE overheads for rlogin, TELNET, and FTP are modest, with a maximal impact of 13% degradation in the worst case; with additional performance optimization, these probably could be reduced. As is shown in the table, performance actually increases (the negative numbers in the table) for rlogin and TELNET when the client is running DTE, because the DTE client passes a DTE UID which the firewall can accept instead of performing costly authentication. This performance increase does not manifest for FTP because the FTP daemon has its own, always invoked, built-in authentication which we did not disable.

Unlike rlogin, FTP, or TELNET, the HTTP service is approximately 50% slower in the worst case. After analysis, we believe this performance decline for HTTP is somewhat artificial, resulting from the low-performance implementation of the HTTP application gateway in the Firewall Toolkit which does a separate *read()* and *write()* system call for each byte transferred. This overstates DTE system call overheads because, in this test, the system spends most of its time dispatching system calls that each do very little work. More recent application gateways, such as Gauntlet's,[12] perform more efficient I/O; we expect that DTE performance for those gateways should approximate the DTE performance for rlogin, TELNET, and FTP.

For NFS, we used Iozone and NFSstones, two widely-used NFS benchmark packages. The Iozone package tests sequential file I/O by writing a 64 MB sequential file in 8 K chunks, then rewinds it, and reads it back (i.e., it measures the number of bytes per second that a system can read or write to a file). We

many more enhancements could be added in future versions.

[11] ZeusBench version 1.0 is copyright Zeus Technology Limited 1996.

[12] Gauntlet is a registered trademark of Trusted Information Systems, Inc.

		Baseline	Percentage Change (%)	
		(n,n,n)	(n,y,n)	(n,y,y)
Iozone	Bytes/Second Written	107,372.50	-4	-20
	Bytes/Second Read	409,430.50	-28	-38
NFSstones	NFSstones/Second	69.83	-18	-38

Figure 5. NFS Test Results (larger numbers indicate better performance)

chose a file size large enough to prevent the cache from dominating the results. NFSstones creates and deletes many directories, then does a variety of file accesses, including writes, sequential reads, and non-sequential reads. Using these results in a formula, it generates a single numeric indicator of relative NFS performance.

As shown by the results in figure 5, performance of writes under NFS is moderately affected by the addition of DTE to the firewall; adding DTE to the server produces a higher impact on performance, with a 20% performance hit. Reads under NFS, however, dominate NFS performance, with a slowdown of 38% when both the firewall and the server are DTE hosts. However, neither the NFS application-level gateway nor the DTE-enhanced NFS server is optimized. Two possible locations for performance degradation in the NFS server are the double mediation necessary because of the primary/auxiliary domain combination and the manipulation of additional file handles needed for DTE mediation in NFS-mounted files. (We believe this last issue dominated the results and can be largely ameliorated.)

5. Related work

DTE firewalls are related most closely to firewall techniques [5, 6], mandatory access controls [2, 3, 4, 7, 10], and type-enforcing systems [4, 13, 14, 16, 20].

There are three fundamental types of firewalls: packet-filtering, circuit gateway, and application gateway. Packet-filtering firewalls allow packets to pass through only if they satisfy a set of filtering rules based on packet direction, physical interface, and a variety of packet fields (e.g., source address, destination address, UDP/TCP port numbers, etc). Circuit gateway firewalls force all TCP connections to go through an intermediary process which performs initial access control and then copies (and perhaps audits) data streams. Application gateway firewalls force data streams penetrating a firewall to be processed on a per-protocol basis by a trustworthy application proxy program that audits protocol usage and possibly screens damaging data. DTE could be added to either packet-filters or circuit gateways; however, we have incorporated DTE into application gateway firewalls because they provide greater opportunities to explore interactions between DTE and individual protocols.

Mandatory access controls, sometimes called "rule-based" controls, are centrally configured by system managers and then enforced on ordinary users and their software. A variety of access control techniques [1, 2, 3, 4, 7, 10] provide centralized control over resources. In general, DTE policies are a proper superset of the DoD lattice model [2] and its integrity variation [3]: DTE can be configured to provide a lattice but also can enforce non-hierarchical security policies such as assured pipelines [4] that drive information through policy-specified pathways of arbitrary connectivity and complexity. DTE also can be configured to provide integrity categories as in [10] and to support the transformation procedures and constrained data items of the Clark/Wilson model [7].

A number of systems have implemented type enforcement, including the Secure Ada Target [4] (later renamed LOCK [14]), Trusted XENIX [16], and DTOS [8], which adds type enforcement to Mach port, task, and virtual memory abstractions. Type enforcement also has been integrated into at least one Internet firewall product, the SCC Sidewinder[13] system, [17] an embedded turnkey system employing a fixed, vendor-supplied access control configuration. DTE [1], an enhanced form of type enforcement, was first demonstrated on an OSF/1 UNIX prototype and later ported to BSD/OS.

6. Future directions

This paper discusses the first phase of our work: a manually-administered DTE firewall prototype. We plan two additional phases to increase the user-friendliness and flexibility of the prototype.

For the second phase, we have formulated simple DTEL modules for organizing policies into more main-

[13] Sidewinder is a trademark of Secure Computing Corporation, Inc.

tainable segments. We also have formulated enhancements that allow DTEL modules to express administrative agreements between network enclaves and to support interactions between compatible but not identical policies. Currently, we are experimenting with loosely-constrained DTEL modules that can be loaded and unloaded while the system runs, at the discretion of the administrator. Since the modules may affect running processes and existing DTE policy elements (in this phase), the administrator must exercise caution in their use. Additionally, we have integrated IP-layer cryptography into the DTE kernel in order to extend DTE protections across unprotected WANs. For this encryption, we chose IPSec in order to provide IPv6 protection mechanisms to IPv4 packets, and are basing our prototype on the Naval Research Laboratory's IPv6/IPSec Software Distribution. As IPv6 becomes more widely available, we will port the DTE network code to use its features.

In the third phase, we are designing facilities for central administration of security policies within enclaves, and for convenient configuration of policy elements that are shared between enclaves. In this phase, each DTE host will boot with a minimal security policy and then contact an enclave-specific Domain and Type Authority (DTA) server to obtain the remainder of its policy, which will be expressed as a set of dynamically loadable DTEL modules. The DTA for an enclave will be responsible for managing consistent and dynamic security policies for hosts within its enclave and also for configuring policies in the DTE firewall and specified hosts to support relationships with external entities.

7. Conclusions

As currently realized, firewall security perimeters are inexpensive but also inflexible and somewhat weak. A central question for practical commercial security is whether the advantages of firewalls can be preserved while adding additional security controls to protect against inside attacks, to protect sensitive data from export, and to fortify servers against corruption from the network. Adding DTE to firewalls appears to address many security concerns because DTE supports role-based policies that relate resource access to individual responsibilities within the enterprise; and because roles provide an intuitive framework for expressing controlled sharing of resources between organizations. We believe the primary question for enhanced-security firewalls is whether useful security features can be added while preserving the functionality of existing applications and protocols, interoperating with non-enhanced systems and programs, imposing minimal performance overheads, and imposing acceptable administrative costs.

This paper reports our results with a prototype DTE firewall system that runs application gateways in DTE domains, exports network services that run in DTE-protected environments either on the firewall itself or on protected servers, and controls interactions with non-DTE systems. In general, our prototype DTE firewall coordinates DTE security protections of communicating endpoints and ensures that clients and corresponding servers execute in the same DTE domains. This relatively simple strategy has allowed us to demonstrate controlled sharing through the firewall, protection of sensitive information, confinement of client applications, and increased protection of network servers from client-based attacks. We have found that security can be increased significantly if DTE runs on the firewall and selected server systems. In addition, running DTE on client systems enables support for user roles and confinement of client applications. For rlogin, TELNET, FTP, HTTP, and mail, we have found no significant decrease in functionality or compatibility. The functionality of the NFS protocol has increased since it previously could not be exported safely. For many protocols, performance is not significantly affected by DTE; we believe that significant performance impacts for NFS and HTTP can be eliminated through optimization techniques. Finally, administrative costs, while still an open issue, appear to be manageable since useful DTE policies can be expressed concisely.

References

[1] L. Badger, D. F. Sterne, D. L. Sherman, and K. M. Walker. A Domain and Type Enforcement UNIX Prototype. In *Usenix Computing Systems Volume 9*, Cambridge, Massachusetts, 1996.

[2] D. E. Bell and L. LaPadula. Secure Computer System: Unified Exposition and Multics Interpretation. Technical Report ESD-TR-75-306, Electronics Systems Division, AFSC, Hanscom AF Base, Bedford, Massachusetts, 1976.

[3] K. J. Biba. Integrity Considerations for Secure Computer Systems. Technical Report ESD-TR-76-372, USAF Electronic Systems Division, Bedford, Massachusetts, 1977.

[4] W. E. Boebert and R. Y. Kain. A Further Note on the Confinement Problem. In *IEEE 1996 Carnahan Conference on Security Technology*, pages 198–203, 1996.

[5] D. B. Chapman and E. D. Zwicky. *Building Internet Firewalls.* O'Reilly and Associates, Inc., 1995.

[6] W. R. Cheswick and S. M. Bellovin. *Firewalls and Internet Security: Repelling the Wily Hacker.* Addison-Wesley, 1994.

[7] D. D. Clark and D. R. Wilson. A Comparison of Commercial and Military Computer Security Policies. In *Proceedings of the 1987 IEEE Symposium on Security and Privacy*, Oakland, California, 1987.

[8] T. Fine and S. E. Minear. Assuring Distributed Trusted Mach. In *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, Oakland, California, 1993.

[9] J. Gosling and H. McGilton. The Java Language Environment: A White Paper. Technical report, JavaSoft, Mountain View, California, May 1996.

[10] S. B. Lipner. Non-Discretionary Controls for Commercial Applications. In *Proceedings of the 1982 IEEE Symposium on Security and Privacy*, Oakland, California, 1982.

[11] National Computer Security Center. *A Guide to Understanding Discretionary Access Control in Trusted Systems*, September 1987.

[12] The Object Management Group, Inc., Framingham, Massachusetts. *The Common Object Request Broker Architecture and Specification*, revision 2.0 edition, July 1995.

[13] R. O'Brien and C. Rogers. Developing Applications on LOCK. In *Proceedings of the 14th National Computer Security Conference*, pages 147–156, Washington D.C., October 1991.

[14] O. S. Saydjari, J. M. Beckman, and J. R. Leaman. LOCK Trek: Navigating Uncharted Space. In *Proceedings of the 1989 IEEE Symposium on Security and Privacy*, Oakland, California, 1989.

[15] D. L. Sherman, D. F. Sterne, L. Badger, S. L. Murphy, K. M. Walker, and S. A. Haghighat. Controlling Network Communication with Domain and Type Enforcement. In *Proceedings of the 1995 National Information Systems Security Conference*, Baltimore, Maryland, 1995.

[16] D. Sterne. A TCB Subset for Integrity and Role-Based Access Control. In *Proceedings of the 15th National Computer Security Conference*, pages 680–696, Baltimore, Maryland, 1992.

[17] D. J. Thomsen. Sidewinder: Combining Type Enforcement and UNIX. In *Proceedings of the 11th Computer Security Applications Conference*, Orlando, Florida, December 1995.

[18] K. W. Walker, D. F. Sterne, M. L. Badger, M. J. Petkac, D. L. Sherman, and K. A. Oostendorp. Confining Root Programs with Domain and Type Enforcement. In *Proceedings of the 6th Usenix Security Symposium*, San Jose, California, 1996.

[19] J. Williams. *Bots and Other Internet Beasties.* Sams.net Publishing, 1996.

[20] S. Wiseman. A Secure Capability Computer System. In *Proceedings of the 1986 IEEE Symposium on Security and Privacy*, Oakland, California, 1986.

: AQ

New Text Document.txt

22 JANUARY 1998

POC: ROME LAB
COMPUTER SYSTEMS BRANCH
ROME, NY 13441-3625